RL-TR-92-297
Final Technical Report
December 1992

CONSTRAINTS LOGIC PROGRAMMING IN KNOWLEDGE-BASED PLANNING DOMAINS

Calspan-UB Research Center

William B. Day

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-92-297 has been reviewed and is approved for publication.

APPROVED: John J. Crowter

JOHN J. CROWTER
Project Engineer

FOR THE COMMANDER: John A Graniero

JOHN A. GRANIERO
Chief Scientist
Command, Control and Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL(C3CA) Griffiss AFB, NY 13441-4505.This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1992	Final Jun 92 - Jul 92

4. TITLE AND SUBTITLE
CONSTRAINTS LOGIC PROGRAMMING IN KNOWLEDGE-BASED PLANNING DOMAINS

5. FUNDING NUMBERS
C - F30602-88-D-0025
Task 0-7-2552
PE - 63728F
PR - 2532
TA - 01
WU - 39

6. AUTHOR(S)
William B. Day

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
CALSPAN-UB Research Center
P.O. Box 400
Buffalo NY 14225

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (C3CA)
525 Brooks Road
Griffiss AFB NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-92-297

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: John J. Crowter/C3CA/(315)330-2073.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)
This report describes an investigation of constraint logic programming languages and their application to knowledge-based planning systems. An overview of constraint logic programming (CLP) languages is presented with specific attention to CLP(R), a Prolog based CLP language. Two planning/scheduling systems are presented and features identified as potential sources of CLP(R) improvements. Finally, two examples are presented.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Artificial Intelligence, Knowledge-Based Planning, Logic Programming, Constraint Logic Programming, Constraint Satisfaction	40
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

1. Introduction

Constraint Logic Programming languages span the void between symbolic and numeric computation. The increased power of expression that these languages present to a programmer expands the field of applications that logic-based, declarative languages can represent. This report is an initial effort to introduce the specific language CLP($\Re$) to the planning and scheduling area of artificial intelligence.

Section 2 of this work is an overview of Constraint Logic Programming, including the motivation and design goals of CLP($\Re$). Short examples are also given in this section to illustrate new and useful features of the language. A description of the CLP($\Re$) architecture is added to complete the system's basic structure. Finally, this section presents three application areas of CLP($\Re$): (1) financial analysis, (2) problems from electrical engineering, and (3) analysis and partial synthesis of truss structures.

Section 3 describes two current planning or scheduling systems. The short summaries that are presented there are intended to emphasize the basic features or to illustrate those characteristics of the systems that are deemed appropriate for CLP($\Re$).

Section 4 discusses two original planning examples that link CLP($\Re$) with planning problems. The codes for the two examples are contained in the appendices. Conclusions for this work are found in Section 5.

2. Constraint Logic Programming

"The class of Constraint Logic Programming (CLP) languages has caused much excitement in computer science because of its conglomeration of such diverse areas as numerical analysis, artificial intelligence, operations research, logic, mathematics, and formal languages" [Cohen 1990]. The combination of logic programming with constraint satisfaction is a natural extension of traditional logic programming languages since the fundamental unification process can be viewed as a special case of constraint satisfaction; i.e., unification of f(X,a) with f(b,Y) is equivalent to solving f(X,a)=f(b,Y).

Constraint Logic Programming (CLP) is a class of logic programming languages that are modeled on rule-based constraint programming. The specific instant CLP($\Re$) uses

the real numbers ($\Re$) as its structure of computation. In particular, the constraints are relations over "real arithmetic terms with uninterpreted functors" [Jaffar 86]. Since all CLP languages are soundly based on a single framework of formal semantics, there are no restrictions in CLP($\Re$) to the Herbrand Universe, to unification, or to allowed equations.

There are several features of CLP($\Re$) that are note-worthy. The most important features are briefly described here and illustrated more fully later. The output of CLP($\Re$) programs is symbolic; thus, it is not only possible to see responses to queries like X=6 or Y=bill, but one may also have X*X = Y+4.

CLP($\Re$) also expands the use of declarative programming. For example, in using an equation of the form X = Y + 4, it is no longer necessary to write two different rules, depending on whether X or Y is instantiated at the time that the equation is invoked. This equation is merely a constraint, and CLP($\Re$) is able to deal with all four possible instantiation cases: neither X nor Y is ground, X is ground, Y is ground, or both X and Y are ground. From the programmer's view, it is only required to state the equation. CLP($\Re$) then makes the appropriate interpretation during execution.

CLP($\Re$) is able to run Prolog programs with no changes. Indeed, one of the design goals of CLP($\Re$) was not only to execute Prolog programs but to have them run as efficiently as normal. This required separating the interpreter into an inference engine and a constraint solver. In running a Prolog program, there should be no need for the constraint solver. Consequently, the inference engine is essentially a Prolog interpreter.

The second fundamental design goal of CLP($\Re$) dictated that only linear constraints would be considered by the constraint solver. This limitation was ameliorated by defining a delay and wakeup condition. Basically, non-linear constraints are delayed until a sufficient number of the variables are instantiated and to make the constraints linear.

The third design goal required CLP($\Re$) to deal with constraints incrementally. This means that in the execution of a query it is not necessary to verify the solvability of an entire set of constraints when a single new constraint is added. Rather it is necessary only to verify that the new constraint is compatible with the old set of constraints.

Other design goals of CLP($\Re$) included a choice of appropriate algorithms for testing the satisficability of systems of constraints, a demand that the system be able to simplify

a system of constraints, and that the constraints be represented internally in a canonical form. These latter objectives deal with the very practical issues of constructing an efficient implementation. They will not be examined further, but their solution will be lauded.

2.1 Constraints versus Assignments

In this section two examples are given to illustrate the increased power of expression available to CLP($\Re$). Both examples involve replacement of the assignment statement ("is") in Prolog with a constraint.

Consider the following predicate which defines the addition of the first two argument:

```
plus(X,Y,Z) :- Z is X+Y.
```

This rule assumes that the variables X and Y have been instantiated at the time the rule is invoked. If either X or Y is not ground or if Z is ground at invocation, a run-time error occurs. In order to make this rule valid in all possible circumstances, it is necessary to replace it with the following set of rules:

```
plus(X,Y,Z) :- not(var(X)), not(var(Y)), not(var(Z)),
                    S is X+Y, Z=S.
plus(X,Y,Z) :- not(var(X)), not(var(Y)), var(Z), Z is X+Y.
plus(X,Y,Z) :- not(var(X)), var(Y), not(var(Z)), Y is Z-X.
plus(X,Y,Z) :- var(X), not(var(Y)), not(var(Z)), X is Z-Y.
```

CLP($\Re$) allows this set of rules to be replaced by this more declarative representation:

```
plus(X,Y,Z) :- Z=X+Y.
```

Here the intention is much clearer: the variables X, Y, and Z are constricted to obey the given equation. It is not necessary for the programmer to described separately the four instantiation cases.

Moreover, it is equally awkward in traditional Prolog to consider the other four possibilities of variable instantiations: two of the three variables are uninstantiated or all three variables are uninstantiated. CLP($\Re$) accepts the single equation and internally considers the separate cases. In CLP($\Re$) the response may be either yes/no (when all variables are instantiated), an assignment (when two of the three variables are instantiated), or an equation (in other cases).

The second example compares the definition of the Fibonacci series in Prolog and in CLP($\Re$).

in Prolog

```
fib(0,1).
fib(1,1).
fib(N,R) :-
  N1 is N-1,
  fib(N1,R1),
  N2 is N-2,
  fib(N2,R1),
```

in CLP($\Re$)

```
fib(0,1).
fib(1,1).
fib(N,R1+R2) :-
  N >= 2,
  fib(N-1,R1),
  fib(N-2,R2).
  R is R1+R2.
```

Both of these programs can answer the question *?-fib(10,F)* with the answer *F=89*, but only the CLP($\Re$) program can answer the question *?-fib(N,89)* with *N=10*. This example illustrates several features of CLP($\Re$). First, the second program is more declarative: to find the Nth Fibonacci number, add the Fibonacci numbers of the (N-1)st and (N-2)nd Fibonacci numbers if N is larger than one. Secondly, the CLP($\Re$) definition allows a reversal of role of the "input" variable and the "output" variable. Although this is common with many other Prolog predicates (compare, the standard definitions of member and append), it is not the intention here. Finally, the CLP($\Re$) version allows arithmetic operations to occur in the arguments of a predicate. This is a radical notion only for Prolog since most procedural languages have generally provided such facilities for procedure calls.

2.2 The CLP($\Re$) Interpreter

Arithmetic terms in CLP($\Re$) are constructed from real constants,variables, and these interpreted functors: +, -, * , /, sin, cos, tan, pow (the power function), min, and max. The CLP($\Re$) interpreter is composed of five parts: (1) the inference engine, (2) the interface between the inference engine and the constraint solver, (3) the equation solver, (4) the inequality solver, and (5) the output module.

The constraint solver itself is composed of the equation and the inequality solvers. The inference engine is basically a Prolog interpreter, which controls the execution of the derivation steps and updates variable bindings. The usual stack mechanisms and symbol tables are maintained here. The interface module's primary purpose is to smooth

the transition between the inference engine and the pieces of the constraint solver. The interface evaluates complex arithmetic expressions and transforms constraints into standard forms.

Constraints that are received by the constraint solver occur in one of three forms: linear equations, linear inequalities, or non-linear equations. Linear equations are sent to the equation solver while linear inequalities are sent to the inequality solver. Non-linear equations are delayed until so many of its variables are ground that the equations become linear. The output module finally converts the internal representations of constraints to a canonical output form.

"The central data structure in the equality solver is a tableau which stores, in each row, a representation of some variables in the form of a linear combination of parametric variables" [Jaffar 1988].

As progress is made with the linear constraints, the involved variables become ground. It is then possible to check if these variables are involved in any of the delayed non-linear constraints. After making appropriate adjustments to the non-linear constraints, a queue is formed of all non-linear constraints that have become linear. These linear constraints are then passed to either the equation solver or the inequality solver before proceeding with the next derivation step of the proof. Chaining effects in which one linearized constraint causes another non-linear constraint to become linear are possible.

A simple bookkeeping device is used to determine when backtracking in the constraint solver is necessary. In general, backtracking in the constraint solver always corresponds to a backtracking point in the inference engine but not vice versa.

Linear equations are stored in parametric form. Thus, a constraint like X = Y, which involves program variables X and Y may be represented as these two parametric equations: X = t and Y = t. In general, any variable will be converted to this parametric form: $X = b + c_1t_1 + c_2t_2 + \ldots + c_kt_k$.

Linear inequalities are solved by the Two-Phase Simplex algorithm. Details of this algorithm are given in [Jaffar 1986].

2.3 Applications of CLP($\Re$)

CLP($\Re$) has been used for an ever increasing number of applications. These three examples are cited to display the diversity: (1) options trading on the stock market, (2) electrical engineering circuit analysis, and (3) truss analysis and partial synthesis of structures in civil engineering.

2.3.1 CLP($\Re$) in Financial Analysis

Options are used to express confidence levels in the intrinsic value of an asset. For example, one may make a call option to buy a stock before some specific date. Should the stock drop very low before the specified date, the owner of the call option would typically purchase the stock at its low point. In the worst case, the stock increases monotonically, and the owner must buy the stock when the specified date arrives.

Two features of options trading have been cited [Lassez 1987] that make it an appropriate application of combining symbolic and numeric computation. They are that traders typically combine (1) their personal, expert heuristics with (2) numeric valuation functions in decision-making.

The indifference of rules in CLP($\Re$) to dictating which variables are input and which variables are output is an attractive characteristic to option trading since it is not necessary to write additional rules for reversing the direction of computation. Furthermore, in CLP($\Re$) it is possible to unite symbolic output and constraints on goals to obtain "what if" analyses.

An interesting feature of option trading is that payoffs for option combinations can be represented by piecewise linear functions. These functions can be expressed in term of two elementary functions: (1) the Heaviside function, and (2) the ramp function, as shown in [Lassez 1986]. The Heaviside function, h(X,Y), is zero if Y < X and one if Y >= X. The ramp function, r(X,Y), is zero if Y < X, and Y-X if Y >= X.

2.3.2 CLP($\Re$) in Electrical Engineering

A second area of applications using CLP($\Re$) has been with some electrical engineering problems [Heintze 1986]. Three categories of these problems are lumped circuits, dig-

ital filtering networks, and electromagnetic fields.

Circuit problems include both analysis and synthesis of analogue circuits, in which the constraints are either local (e.g., Ohm's law for a resistor) or global (e.g., Kirchoff's law in nodal analysis). Specific examples of these problems are analysis of steady-state RLC circuits and diode components in RLC circuits. Bipolar junction transistors for amplifier and logic circuits have also been analyzed and designed.

In the area of digital signal flow, the simulation of linear shift-invariant digital system with the independent variable time have been successfully modeled in terms of linear signal-flow graphs. In these graphs signal values at the nodes at successive time steps are simulated by summing constraints for each node and by collecting branch equations. In CLP($\Re$) one needs only to declaratively state the equations while the interpreter decides the particular method for solving the equations.

The final category of electrical engineering problems involves numerical, approximate solutions for the pertinent partial differential equations of electromagnetic fields. A finite difference approximation is the common approach for these problems. The examples of this category illustrate the elegance of CLP($\Re$) in using finite difference methods by its concise representations and its use of the constraint collection mechanism.

2.3.3 CLP($\Re$) in Truss Structures

Analysis and synthesis problems are common to all areas of engineering. Recently [Lakmazaheri 1989] has shown that constraint logic programming is suitable for the analysis and partial synthesis of truss structures. Typically these two problems of analysis and synthesis are considered antithetical in engineering; however, a CLP($\Re$) representation is able to use only one representation for the two. The particular query dictates the appropriate form.

This work on truss structures has shown that the technique of local propagation is inadequate for satisfying simultaneous constraints when the associated constraint graph contains cycles. But in the constraint-based approach the constraints can be viewed as relations rather than functions; hence, there is no explicit commitment to a specific variable being either input or output.

The constraints associated with structural components are classified by these categories: (1) truss components, which satisfy a two or three dimensional form of the vector equation KD=F with K an n by n (n=2 or 3) stiffness matrix, D the n by 1 displacement vector at the two end nodes and F the n by 1 force vector at the two end nodes; (2) support components of the type pin support, rollerX support, or rollerY support; and (3) load components, each of which is associated with a force vector and a displacement vector.

The position and displacement vectors are functions, and the nodes' force histories are lists. Nodes in two dimensions are terms of this form:

n(p(Px,Py), d(Dx,Dy), H)

where Px and Py are the x and y coordinates of the position vector, Dx and Dy are the x and y coordinates of the displacement vector, and H is the node force history list.

One can then describe general rules for the local nodal behaviors. This is analogous to Ohm's law in electrical circuits. For example, this rule

A valid node exists if its coordinates are valid.

is represented by this CLP($\Re$) rule

node(n(p(Px,Py),_,_)) :- ordinate(Px), ordinate(Py).

Since the behavior of a structural model is derived from the behavior of its structural components, rules can be added that insure the satisficability of the constraints associated with the structural components.

Four examples are presented in [Lakmazaheri 1989]. The first example illustrates the capability of the constraint logic program to perform numerical analysis of a statically indeterminate truss structure. The second example shows the capability of the program to perform symbolic computation for structural analysis. The third example gives a partial synthesis of a simple truss structure using the same program. Finally, the last example demonstrates the applicability of the program for solving more complex problems.

The main contribution of this work is the development of a general and declarative constraint-based formulation for the analysis and partial synthesis of truss structures using the framework of constraint logic programming methodology. A single and uniform framework for modeling engineering design problems facilitates the integration of

different design activities such as synthesis, analysis, and evaluation.

Currently [Lakmazaheri 1992] is examining two strategies to improve the processing time of constraint-based engineering problems by using distributed processing through problem decomposition. The set of connectivity constraints then ensures displacement compatibility and force equilibrium on the connections between each pair of substructures. A study of 5000 elements has shown a speed-up factor of 6 is achieved when the structure is decomposed into sixteen substructures.

3. Planning

Constraints are a familiar part of planning and scheduling problems. The choices that are made in such a design involve relations (constraints) among objects of the domain of discourse. The use of constraints in artificial intelligence was proposed in [Steele 1980] and [Sussman 1980].

This section describes briefly two current planning or scheduling systems. The short summaries that are presented here are intended to emphasize the basic features or to illustrate those characteristics of the systems that are deemed appropriate for CLP($\Re$).

3.1 OPIS

The difficulties in factory scheduling, as defined in [Smith 1990] are caused by the complexity of assigning shared resources to competing processes and by the unpredictability of maintaining a stable schedule of good quality within acceptable time limits. OPportunistic Intelligent Scheduling (OPIS) is an incremental scheduler that attacks these difficulties through the blackboard paradigm. OPIS uses characteristics of the current solution constraints to focus the attention of the top-level manager of the knowledge sources (KSs). On each control cycle OPIS combines constraint propagation with consistency maintenance.

OPIS uses dynamic decomposition to manage the complexity problem. This requires heuristic guidance about the decomposition and where to direct the search effort. Two decomposition types used in OPIS are resource-based and order-based. As implied by the naming, a resource-based decomposition emphasizes a specific resource and stress-

es optimal resolution of conflicts over that resource. In contrast, order-based decomposition emphasizes a particular customer's order and is directed by the optimal resolution of conflicts to produce that order.

The architecture of OPIS is suitable for all forms of scheduling. In this framework knowledge sources are either analysis or scheduling knowledge sources. Schedule KSs post updates for adjusting the incrementally maintained representation of the current, incomplete schedule. Updates are one of these three kinds: (1) incomplete hypotheses (processes that have yet to be scheduled), (2) elementary conflicts, and (3) opportunities available due to loosening of the constraints.

Analysis KSs are used within the top-level control cycle, which consists of event selection, event analysis, action section, and action execution. Event selections are based on aggregating primitive events and establishing priorities. Event analysis examines capacity and conflict constraints. Capacity constraints are used to estimate anticipated contention by identifying possible bottlenecks. Conflict constraints are measured by duration of the conflict, the number of order involved, projected lateness, etc.

The two primary schedulers of OPIS are the order scheduler and the resource scheduler. The order scheduler uses beam search on resource assignment. The resource scheduler is an iterative dispatch-based methodology. Other schedulers that have been tried simply push scheduled execution times forward (the right shift scheduler) or exchange the remaining part of an incomplete plan with another order's schedule in order to optimize the two plans (demand swap).

3.2 Deadline-Coupled Real-Time Planning

Real-time planning is an important class of planning problems that contain a mixture of symbolic and numeric computations. In real-time planning it is necessary to spend time not only planning and acting but also reasoning about the time remaining before reaching a deadline. This latter reasoning is traditionally done through meta-reasoning, and its timing requirements are frequently ignored. In [Kraus 1990] an accounting is made for all planning and acting times. This is done using step-logics, in which the fundamental unit (a step) represents the time taken to draw a single inference. At the ith step, the inference engine uses observations of the external environment as input, together with all non-contradictory deductions drawn in the previous i-1 steps.

Until a plan is fully developed, the reasoner works on the current partial plan, which consists of an ordered list of triples, each of which is (1) a set of associated preconditions, (2) an action, and (3) a set of following results. This representation is able to distinguish between actual facts and the facts within the context of the partial plan: i.e., the difference between a robot being actually located at position P and the planning precondition that the robot is located there.

This approach also allows execution of the partially developed plan to begin before a complete plan has been constructed. While the action occurs, planning may continue. During acting and planning, the system maintains a Working Estimate of Time (WET) remaining before the deadline. Furthermore, each action has a timing interval associated with it. During this interval, the predicate (action) is asserted to be true.

4. Planning Examples in CLP($\Re$)

4.1 Departmental Scheduling of Courses

The first scheduling example is a program for constructing a department's schedule of course offerings for the next term. The program is somewhat restricted, but does contain many of the features needed to assess the potential of CLP($\Re$) as an adequate representation of scheduling problems.

The entire program is divided into three parts: (1) the rules of inference, (2) constraints that must be met, and (3) instructors' preferences, which will be satisfied if possible. The inference rules initially construct a schedule that meets all constraints and then attempts to accommodate each of the preferences.

It is assumed that there is only one section for each of eighteen courses and that a particular instructor has already been assigned to each course. These facts are given in this form: course(560,jelks). The department controls only three classrooms (room1, room2, room3). All classes begin on the hour and last fifty minutes The earliest class begins at 8 a.m., and the latest class begins at 2 p.m. Thus, there are twenty-one time/room combinations available for the eighteen courses. No allowance is made for Monday-Wednesday-Friday or any other form of scheduling. One can assume that each class needs its room every day.

Two types of constraints are made on the schedule. First, no instructor can teach two courses at the same hour. Secondly, certain pairs of courses cannot be scheduled at the same hour since such an alignment will prevent departmental majors from taking both classes. The second type of constraint is represented by facts like this: time_clash(400,520).

The initial scheduling recursively invokes the predicate "schedule/3", which selects the next_candidate. This is an order scheduler in which the order is dictated by the hour, with a numerical sub-ordering of the three rooms. This scheduler can be modified to a resource scheduler. For example, one can impose a priority measure based on the course number or on the particular instructor. An alternative is to construct the constraint graph of the "time_clash" predicate and let the resource scheduler select the most heavily constrained course/instructor first. In the program of Appendix A, the course/instructor selected for the current candidate time/room by the order scheduler is based on a top-to-bottom scan of the "course" facts.

For a large scale system, it may be more appropriate to use a priority queue of "course" facts. Standard backtracking methods are available for constructing alternative schedules. In the Appendix A example, a time/room is left vacant if no compatible course/instructor is available. These unscheduled exceptions are reported to the user when the Initial Schedule is presented.

After the initial schedule is completed, adjustments are attempted for each instructor's preferences that are not satisfied in the Initial Schedule. These preferences may take many forms. An instructor may wish to avoid certain hours; e.g., no class after noon. She may wish to avoid a particular room assignment. He may wish to avoid teaching back-to-back classes. For illustration, the Appendix A program uses only preferences for avoiding particular class hours.

Many priority orders of the preferences are available; viz., by seniority, by multiplicity, by professorial rank, etc. Each unachieved preference is tried only once in the following fashion. If (1) there is an available time/room in the current schedule, (2) no constraints will become invalid , and (3) no other achieved preference is negated, then the course/instructor is moved to the new time/room, and the old time/room becomes free. If any one of these conditions is not met, the unfulfilled preference in noted (in the output), but it is not reconsidered.

4.2 Where Is the Zebra?

The second example was originally available on network services and solved by Daniel Ligett. As stated, one is given five house in a line. Each house has one of five colors (green, red, blue, yellow, ivory), shelters one of five nationalities (English, Spanish, Ukrainian, Japanese, Norwegian), has one of five pets (dog, horse, zebra, fox, snails), stocks one of five liquids (water, tea, coffee, orange juice, milk), and breathes one of five cigarettes (Kools, Chesterfields, Parliaments, Winstons, Lucky Strikes).

With no constraint, there are 5x5x5x5x5 different combinations that can occupy the first house, 4x4x4x4x4 combinations for the second house, 3x3x3x3x3 combinations for the third house, 2x2x2x2x2 combinations for the fourth house, and one remaining combination for the fifth house. Thus, there are 3125x1024x243x32 (about 24 billion) total possible combinations.

Fourteen constraints are given to limit the search space in locating the house of the zebra. All the restrictions are shown in the complete program which is given in Appendix B. Eight of the constraints pair two of the five attributes: e.g., the Japanese smokes Parliaments. Three of the constraints use the "next_to" predicate: e.g., the house with chesterfields is next to the house of the fox. One restriction is even more specific, using the predicate "is_to_the_right_of" rather than "next_to." One constraint describes the first house (counting from left to right), and one describes the middle house.

With the fourteen constraints, one finds only one solution: the zebra resides in the fifth house. If the constraint that links the Japanese with Parliaments is dropped, one finds nine solutions, and the zebra can be found in any of the houses except the second house. If the constraint that connects snails and Winstons is also dropped, one finds sixty-four solutions. If the constraint between the green house and coffee is dropped, one has 344 solutions. The case of one solution and the case of nine solutions are given in Appendix B.

In general, one cannot expect always to find ten solutions by dropping one of the fourteen constraints. For example, if the only missing restraint is the one that requires the house of the Chesterfields to be next to the house with the fox, then one find just two solutions, and the zebra is found in either the first or the fifth house.

The original solution has been modified by adding the predicate "member_restrictions" in order to guarantee no duplications of any of the five attributes. This is implied in the statement of the problem but did not actually happen except in the case of the unique solution.

A variation of the modified program also appears in Appendix B. In this second form the five values of each of the five attributes is coded numerically. For example, red=1, blue=2, green=3, yellow=4, ivory=5 for the color attribute. Also the uniqueness constraints, which are invoked with the "member_restrictions" predicate are replaced by two predicates which use numeric constraints. The first predicate, "member_restrictions," checks that each value assumed by each of the five attributes lies in the closed interval from one to five. The second predicate, "column_restrictions," checks that no value is duplicated. If each numeric value is assumed only once, then their sum of all five houses for each attribute must be fifteen.

5. Conclusions

In this report a discussion of constraint logic programming has been presented. This included (1) features of the CLP($\Re$) language that extend logic programming to linear equations and inequalities over the reals, (2) the architecture of CLP($\Re$), and (3) application of CLP($\Re$) to financial analysis, electrical engineering, and truss structures. Two planning/ scheduling systems were also examined to show that such systems could contain features that would be appropriately represented with CLP($\Re$). Finally, two small examples were constructed, a departmental course scheduler and a reformulation of "where's the zebra" problem. The scheduler illustrates CLP($\Re$)'s expressive power in planning/scheduling problems and in general problem-solving which involves symbolic constraints. The "zebra" puzzle shows the versatility of CLP($\Re$) with representing symbolic and numeric constraints.

Future work in this area will be directed to examining how the OPIS schedulers and the deadline-directed real-time scheduler can be expressed in CLP($\Re$). Hopefully, these examinations will provide an impetus for using CLP($\Re$) in other areas of planning and scheduling.

6. References

[Cohen 1990]
Cohen, Jacques, "Constraint Logic Programming Languages," Communications of the ACM, July 1990, 52-68.

[Heintze 1986]
Heintze, N., S. Michaylov, and P. Stuckey, "CLP($\Re$) and Some Electrical Engineering Problem," Proc. of 4th Int. Conf. on Logic Programming, J-L. Lassez (Ed.), MIT Press, 1987.

[Jaffar 1986]
Jaffar, J. and S. Michaylov, Methodology and Implementation of a CLP System, Proc. 4th Int. Conf. on Logic Programming, J-L. Lassez (Ed.), MIT Press, 1987.

[Jaffar 1988]
Jaffar, J., S. Michaylov, P. J. Stuckey, and R. H. C. Yap, The CLP($\Re$) Language and System, Constraint & Languages Workshop, April 1988.

[Kraus 1990]
Kraus, S., M. Nirkhe, and D. Perlis, "Deadline-Coupled Real-Time Planning," Proc. Workshop on Innovative Approaches to Planning, Scheduling and Control Workshop, K. Sycara (Ed.), San Diego, CA, November 1990, 100-108.

[Lakmazaheri 1989]
Lakmazaheri, S. and W. J. Rasdorf, "Constraint Logic Programming for the Analysis and Partial Syntheses of Truss Structures," Artificial Intelligence for Engineering Design, Analysis, & Manufacturing, 3 (1989), 157-173.

[Lakmazaheri 1992]
Lakmazaheri, S, "Sequential versus Distributed Constraint-Based Approach to Structural Design Automation: A Comparative Study," Proc. 8th Conf. on Computing in Civil Engineering, Dallas, TX, June 1992, to appear.

[Lassez 1987]
Lassez, C., K. McAloon, and R. Yap, "Constraint Logic Programming and Option Trading," IEEE Expert, 2, Fall 1987, 42-50.

[Smith 1990]
Smith, S. F., P. S. Ow, N. Muscettola, J-Y. Potvin, and D. C. Matthys, "OPIS: An Integrated Framework for Generating and Revising Factor Schedules," Proc. Workshop on Innovative Approaches to Planning, Scheduling and Control Workshop, K. Sycara (Ed.), San Diego, CA, November 1990, 497-507.

[Steele 1980]
Steele, G. L., The Implementation and Definition of a Computer Programming Language Based on Constraints, Ph.D. dissertation, Dept. of EE and CS, MIT, Cambridge, Mass. (Artificial Intelligence Tech. Rep. AITR 595), August 1980.

[Sussman 1980]
Sussman, G. J., and G. L. Steele, "Constraints: A Language for Expressing Almost-Hierarchical Descriptions," Artificial Intelligence, 14 (1980), 1-19.

APPENDIX A

```
/*****************************************************************/
/*    Scheduling 18 courses (with instructors) into three classrooms,    */
/*        between 8 a.m. and 2 p.m.                                      */
/*    Facts are in files: prog/c_and_f2.clpr, prog/pref2.clpr            */
/*****************************************************************/

/*****************************************************************/
/*                ----------RULES-----------                        */
/*  These rules                                                        */
/*            go,schedule,present,time_room, constraints_valid,      */
/*            double, overload, and course_clash                     */
/*  are used to construct the initial schedule, which honors departmental */
/*  requirements that certain pairs of courses cannot be taught at the */
/*  same hour since this would prevent majors from taking more than one */
/*  of the courses.                                                    */
/*****************************************************************/

go :- schedule ( [], 7, room3).

schedule( L, Told, Rold) :-
                        next_candidate(Told,Rold,Tnew,Rnew,L,Lnew),
                        schedule( Lnew, Tnew, Rnew).
schedule( L, Told, Rold) :-
                        reverse(L,Lrev),
                        exhibit(Lrev).

next_candidate(Told,Rold,Tnew,Rnew,L,Lnew) :-
                        time_room(Told, Rold, Tnew, Rnew),
                        course (N, P),
                        constraints_valid( [N,P,Tnew,Rnew], L),
                        append( [ [N,P,Tnew,Rnew] ], L, Lnew).

next_candidate(Told,Rold,Tnew,Rnew,L,L) :-
                        time_room(Told, Rold, Tnew, Rnew).
```

```
exhibit(L) :-
                    tell('prog/schedule.txt'),nl,nl,
          write('************************************'),
                    write('Initial Schedule for CSE Classes'), nl,
                    present(L),
                    adjust_pref(L, Ladj),
                    nl,nl,write('With the exceptions listed above, '),
                    write('all preferences have been met.'),nl,nl,nl,
          write('************************************'),
                    write('Adjusted Schedule for CSE Classes'),nl,
                    present(Ladj).

time_room(14, room3, Tnew, Rnew) :-
                    fail.
time_room(Told, room3, Tnew, room1) :-
                    not(Told = 14), Tnew = Told +1.
time_room(Told, room1, Told, room2).
time_room(Told, room2, Told, room3).

constraints_valid( [N,P,T,R], L ) :-
                    not( member( [N,P,_,_], L ) ),
                    not( double(  [N,P,T,R], L ) ),
                    not( overload( [N,P,T,R], L ) ).

double( [N,P,T,R], [ [N1,P1,T1,R1] | Tail ] ) :- T=T1, P=P1.
double( [N,P,T,R], [ H | Tail ] ) :- double( [N,P,T,R], Tail ).

overload( [N,P,T,R], [ [N1,P1,T1,R1] | Tail] ) :-
                                        T = T1, course_clash(N,N1).
overload( [N,P,T,R], [ H | Tail] ) :- overload( [N,P,T,R], Tail).
overload( [N,P,T,R], [] ) :- fail.

course_clash(A,B) :- time_clash(A,B).
course_clash(A,B) :- time_clash(B,A).
```

```
present( [ [N,P,T,R] | Tail ] ) :-
                                write(N), write('   '),
                                write(T), write('   '),
                                write(R), write('   '),
                                write(P), nl,
                                present(Tail).

/*****************************************************************/
/*      Rules for Adjusting the Initial Schedule to              */
/*      account for Instructors' Time Preferences                */
/*      TVL is the list of Time Violated preferences             */
/*****************************************************************/

adjust_pref(L,Ladj) :- times_violate(L,[],TVL),
                                adjust_times(L,Ladj,TVL).

times_violate(Lall,Lin,Lout) :-
                                time_noz(P,T),
                                not(member( [_,P,T,_], Lin)),
                                   member( [_,P,T,_], Lall),
                                append( [ [_,P,T,_] ], Lin, Lapp),
                                times_violate(Lall,Lapp,Lout).
times_violate(Lall,Lin,Lout) :-
                                reverse(Lin,Lout).

adjust_times(L,Ladj,TVL) :-
                                car(TVL,[_,P,T,_]),
                                member( [N,P,T,R], L),
                                reschedule( [N,P,T,R], L, Lnew),
                                cdr(TVL,Y), TVLnew = Y,
                                adjust_times(Lnew,Ladj,TVLnew).
adjust_times(L,L,[]).
```

```
reschedule( [N,P,T,R], L, Lnew) :-
                                find_open_times(L,Lavail),
                                member([_,_,Topen,Ropen], Lavail),
                                remove( [N,P,T,R], L, Lrem),
                              check_new_place(N,P,T,R,Topen,Ropen,Lrem,Lnew).

find_open_times(L,Lavail) :-
                                all_times([],7,room3,Las),
                                setdiff(Las,L,Lavail).


all_times(L,Told,Rold,Las) :-
                                time_room(Told,Rold,Tnew,Rnew),
                                append( [ [_,_,Tnew,Rnew] ],L,Lnew),
                                all_times(Lnew,Tnew,Rnew,Las).
all_times(L,Told,Rold,Lrev) :-
                                reverse(L,Lrev).

check_new_place(N,P,T,R,Topen,Ropen,L,Lnew) :-
                                constraints_valid( [N,P,Topen,Ropen], L),
                                not(new_conflict( [N,P,Topen,Ropen])),
                                append( [ [N,P,Topen,Ropen] ],L,Lnew), !.
check_new_place(N,P,T,R,Topen,Ropen,L,Lnew) :-
                                nl,write('>>>>>    '),
                                write('Unable to accommodate the preference of '),
                                write(P),write(' for no class at '),write(T),
                                write('.'),nl,
                                append( [ [N,P,T,R] ],L,Lnew).

new_conflict( [N,P,T,R]) :-
                                time_noz(Pno,Tno), P=Pno, T=Tno.
```

```
/*****************************************************************************/
/*              STANDARD USEFUL RULES                                        */
/*****************************************************************************/

car([[N,P,T,R]|Tail],[N,P,T,R]).
cdr([[N,P,T,R]|Tail],Tail).

append([],L,L).
append( [X|Y], L2, [X|L3] ) :-
                                append(Y,L2,L3).

reverse([],[]).
reverse([H|T],L) :-             reverse(T,Trev), append(Trev,[H],L).


member( [N,P,T,R], [ [N,P,T,R] | _] ).
member( [N,P,T,R], [ H | Trail ] ) :-
                                member( [N,P,T,R], Trail).

remove( [N,P,T,R], [ [N,P,T,R] | Tail ], Tail) :- !.
remove( [N,P,T,R], [ H | Tail], [H | Ldel]) :-
                                remove( [N,P,T,R], Tail, Ldel).

setdiff([],B,[]).
setdiff( [ [_,_,T,R] |A2],B,C) :-
                                member( [_,_,T,R],B), setdiff(A2,B,C).
setdiff([ [_,_,T,R]|A2],B,[ [_,_,T,R] |C2]) :-
                                not(member([_,_,T,R],B)), setdiff(A2,B,C2).
```

```
/* These are the course numbers and the assigned instructor.                */

course(350,deng).
course(324,kabuki).
course(360,elalemain).
course(400,faisal).
course(520,han).
course(624,baker).
course(660,kabuki).
course(200,amin).
course(220,baker).
course(405,govel).
course(505,govel).
course(524,han).
course(530,ijssel).
course(533,amin).
course(560,jelks).
course(561,jelks).
course(618,deng).
course(630,faisal).

/* These are the restrictions on scheduling courses at the same hour.      */
/* Ultimately in the rules each combination is forbidden.                   */

time_clash(200,350).
time_clash(220,350).
time_clash(324,350).
time_clash(324,360).
time_clash(360,400).
time_clash(400,520).
time_clash(405,530).
time_clash(505,533).
time_clash(520,560).
time_clash(524,561).
```

```
time_clash(618,624).
time_clash(624,660).
time_clash(630,660).


/* These are instructors' preferences, constraints which can be relaxed.      */
/*      time_noz(A,B) means instructor A teaches at B o'clock.                */
/*      room_noz(A,B) means instructor teaches in room B.                     */
/*      adj_noz(A) means instructor has adjacent or back-to-back classes.     */
/* Ultimately the rules will deny all three of these preferences.             */

time_noz(amin,8).
time_noz(amin,14).
time_noz(kobuki,12).
time_noz(faisal,10).
time_noz(govel,10).
time_noz(govel,11).
time_noz(govel,12).
time_noz(ijssel,8).
time_noz(jelks,8).
time_noz(jelks,9).

room_noz(deng,r3).
room_noz(ijssel,r1).

adj_noz(baker).
adj_noz(deng).
adj_noz(elalemain).
adj_noz(han).
```

APPENDIX B

THE ORIGINAL VERSION OF "WHERE'S THE ZEBRA"

```
/* Each house is represented by a structure of the form:
        house(Colour, Nationality, Pet, Drink, Cigarette)
   Puzzle is currently missing
        next_to(house(_,_,_,_,chesterfields), house(_,_,fox,_,_), Houses)  */

puzzle :-
        houses(Houses),
        member(house(red, english, _, _, _), Houses),
        member(house(_, spanish, dog, _, _), Houses),
        member(house(_, ukranian, _, tea, _), Houses),
        member(house(_,_,snails,_,winstons), Houses),
        member(house(green,_,_,coffee,_), Houses),
        member(house(_,japanese,_,_,parliaments), Houses),
        right_of(house(green,_,_,_,_), house(ivory,_,_,_,_), Houses),
        member(house(yellow, _, _, _, kools), Houses),
        Houses = [_, _, house(_, _, _, milk, _), _,_],
        Houses = [house(_, norwegian, _, _, _)|_],
        next_to(house(_,_,_,_,kools), house(_,_,horse,_,_), Houses),
        member(house(_, _, _, orange_juice, lucky_strikes), Houses),
        next_to(house(_,norwegian,_,_,_), house(blue,_,_,_,_), Houses),
        member_restrictions(Houses),
        print_houses(Houses),
        nl,nl,write('****************************************'),nl,fail.

 houses([
        house(_, _, _, _, _),
        house(_, _, _, _, _),
        house(_, _, _, _, _),
        house(_, _, _, _, _),
        house(_, _, _, _, _)
 ]).
```

```
member_restrictions(Houses) :-
        member(house(green,_,_,_,_), Houses),
        member(house(red,_,_,_,_), Houses),
        member(house(blue,_,_,_,_), Houses),
        member(house(yellow,_,_,_,_), Houses),
        member(house(ivory,_,_,_,_), Houses),
        member(house(_,english,_,_,_), Houses),
        member(house(_,spanish,_,_,_), Houses),
        member(house(_,ukranian,_,_,_), Houses),
        member(house(_,japanese,_,_,_), Houses),
        member(house(_,norwegian,_,_,_), Houses),
        member(house(_,_,horse,_,_), Houses),
        member(house(_,_,dog,_,_), Houses),
        member(house(_,_,zebra,_,_), Houses),
        member(house(_,_,fox,_,_), Houses),
        member(house(_,_,snails,_,_), Houses),
        member(house(_,_,_,orange_juice,_), Houses),
        member(house(_,_,_,milk,_), Houses),
        member(house(_,_,_,coffee,_), Houses),
        member(house(_,_,_,tea,_), Houses),
        member(house(_,_,_,water,_), Houses),
        member(house(_,_,_,_,kools), Houses),
        member(house(_,_,_,_,chesterfields), Houses),
        member(house(_,_,_,_,lucky_strikes), Houses),
        member(house(_,_,_,_,parliaments), Houses),
        member(house(_,_,_,_,winstons), Houses).


right_of(A, B, [B, A | _]).
right_of(A, B, [X | Y]) :- right_of(A, B, Y).

next_to(A, B, [A, B | _]).
next_to(A, B, [B, A | _]).
next_to(A, B, [X | Y]) :- next_to(A, B, Y).
```

```
member(X, [X|Y]).
member(X, [A|B]) :- member(X, B).

print_houses([A|B]) :-  tell('prog/zout.txt'),
        print(A),nl,
        print_houses(B).
print_houses([]).
```

The Output When All 14 Restriciton Are Included

```
house(yellow, norwegian, fox, water, kools)
house(blue, ukranian, horse, tea, chesterfields)
house(red, english, snails, milk, winstons)
house(ivory, spanish, dog, orange_juice, lucky_strikes)
house(green, japanese, zebra, coffee, parliaments)

**************************************
```

The Output When Only 13 Restriciton Are Included
Missing: house(_,japanese,_,_,parliaments)

house(yellow, norwegian, fox, water, kools)
house(blue, ukranian, horse, tea, chesterfields)
house(red, englisb snails, milk, winstons)
house(ivory, spanish, dog, orange_juice, lucky_strikes)
house(green, japanese, zebra, coffee, parliaments)

****

house(yellow, norwegian, fox, water, kools)
house(blue, ukranian, horse, tea, chesterfields)
house(red, english, zebra, milk, parliaments)
house(ivory, spanish, dog, orange_juice, lucky_strikes)
house(green, japanese, snails, coffee, winstons)

****

house(yellow, norwegian, zebra, water, kools)
house(blue, ukranian, horse, tea, chesterfields)
house(red, english, fox, milk, parliaments)
house(ivory, spanish, dog, orange_juice, lucky_strikes)
house(green, japanese, snails, coffee, winstons)

****

house(yellow, norwegian, fox, water, kools)
house(blue, ukranian, horse, tea, chesterfields)
house(red, english, snails, milk, winstons)
house(ivory, japanese, zebra, orange_juice, lucky_strikes)
house(green, spanish, dog, coffee, parliaments)

****

house(yellow, norwegian, zebra, water, kools)
house(blue, ukranian, horse, tea, parliaments)
house(red, english, snails, milk, winstons)

house(ivory, japanese, fox, orange_juice, lucky_strikes)
house(green, spanish, dog, coffee, chesterfields)

****

house(yellow, norwegian, zebra, water, kools)
house(blue, japanese, horse, orange_juice, lucky_strikes)
house(red, english, snails, milk, winstons)
house(ivory, ukranian, fox, tea, parliaments)
house(green, spanish, dog, coffee, chesterfields)

****

house(yellow, norwegian, fox, water, kools)
house(blue, ukranian, horse, tea, chesterfields)
house(ivory, spanish, dog, milk, parliaments)
house(green, japanese, snails, coffee, winstons)
house(red, english, zebra, orange_juice, lucky_strikes)

****

house(yellow, norwegian, fox, water, kools)
house(blue, ukranian, horse, tea, chesterfields)
house(ivory, japanese, snails, milk, winstons)
house(green, spanish, dog, coffee, parliaments)
house(red, english, zebra, orange_juice, lucky_strikes)

****

house(yellow, norwegian, zebra, water, kools)
house(blue, ukranian, horse, tea, parliaments)
house(ivory, japanese, snails, milk, winstons)
house(green, spanish, dog, coffee, chesterfields)
house(red, english, fox, orange_juice, lucky_strikes)
****

THE NUMERICAL VERSION OF "WHERE'S THE ZEBRA"

```
/*  Each house is represented symbolically by a structure of the form:          */
/*                  house(Color, Nationality, Pet, Drink, Cigarette)            */

/*                                          TRANSLATIONS
COLORS          NATIONALITIES       PETS        DRINKS      SMOKES
1=red           1=english           1=dog       1=water     1=kools
2=blue          2=spanish           2=horse     2=tea       2=chesterfields
3=green         3=ukranian          3=zebra     3=coffee    3=parliaments
4=yellow        4=japanese          4=fox       4=orange_j  4=winstons
5=ivory         5=norwegian         5=snails    5=milk      5=lucky_ strike
*/

/*Missing constraint of japanese and parliaments: member(house(_,4,_,_,3),Houses)*/

puzzle :-
        houses(Houses),
        member(house(1, 1, _, _, _), Houses),
        member(house(_, 2, 1, _, _), Houses),
        member(house(_, 3, _, 2, _), Houses),
        member(house(_,_,5,_,4), Houses),
        member(house(3,_,_,3,_), Houses),
        right_of(house(3,_,_,_,_), house(5,_,_,_,_), Houses),
        member(house(4, _, _, _, 1), Houses),
        Houses = [_, _, house(_, _, _, 5, _), _,_],
        Houses = [house(_, 5, _, _, _)|_],
        next_to(house(_,_,_,_,2), house(_,_,4,_,_), Houses),
        next_to(house(_,_,_,_,1), house(_,_,2,_,_), Houses),
        member(house(_, _, _, 4,5), Houses),
        next_to(house(_,5,_,_,_), house(2,_,_,_,_), Houses),
        member_restrictions(Houses),
        column_restrictions(Houses),
        print_houses(Houses),
```

```
        nl,nl,write('*****************************************').nl.fail.
houses([
        house(_, _, _, _, _),
        house(_, _, _, _, _),
        house(_, _, _, _, _),
        house(_, _, _, _, _),
        house(_, _, _, _, _)
]).

member_restrictions(Houses) :-
                        member(house(C,N,P,D,S), Houses),
                        1 <= C, C <= 5,
                        1 <= N, N <= 5,
                        1 <= P, P <= 5,
                        1 <= D, D <=5,
                        1 <= S, S <= 5, fail.
member_restriction(Houses).

column_restrictions(Houses) :-
                        Houses = [  house(C1,N1,P1,D1,S1),
                                    house(C2,N2,P2,D2,S2),
                                    house(C3,N3,P3,D3,S3),
                                    house(C4,N4,P4,D4,S4),
                                    house(C5,N5,P5,D5,S5) ],
                        C1+C2+C3+C4+C5=15,
                        N1+N2+N3+N4+N5=15,
                        P1+P2+P3+P4+P5=15,
                        D1+D2+D3+D4+D5=15,
                        S1+S2+S3+S4+S5=15.

right_of(A, B, [B, A | _].
right_of(A, B, [X | Y ] :- right_of(A, B, Y).

next_to(A, B, [A, B | _]).
```

```
next_to(A, B, [B, A | _]).
next(to(A, B, [X | Y]) :- next_to(A, B, Y).

member(X, [X|Y]).
member(X, [A|B]) :- member(X, B).

print_houses([A|B]) :-
                    tell('prog/zout.txt'),
                    print(A), nl,
                    print_houses(B).
print_houses([]).
```

U.S. GOVERNMENT PRINTING OFFICE

MISSION

OF

ROME LABORATORY

Rome Laboratory plans and executes an interdisciplinary program in research, development, test, and technology transition in support of Air Force Command, Control, Communications and Intelligence (C^3I) activities for all Air Force platforms. It also executes selected acquisition programs in several areas of expertise. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. In addition, Rome Laboratory's technology supports other AFSC Product Divisions, the Air Force user community, and other DOD and non-DOD agencies. Rome Laboratory maintains technical competence and research programs in areas including, but not limited to, communications, command and control, battle management, intelligence information processing, computational sciences and software producibility, wide area surveillance/sensors, signal processing, solid state sciences, photonics, electromagnetic technology, superconductivity, and electronic reliability/maintainability and testability.